

## Claims

1. A method for efficiently shifting operands for processing, the method comprising:
  - a) receiving an operand;
  - 5 b) determining a shift value to be applied to the operand at a first stage;
  - c) shifting the operand by the shift value;
  - d) outputting the shifted operand to a second stage;
  - e) repeating b), c), and d) for successive stages until the operand is shifted for processing; and
  - 10 g) processing the operand.
2. The method of claim 1, wherein the operands have a bit width of 32 bits.
3. The method of claim 1, wherein the method operation of processing the operand includes,
  - 15 aligning a least significant bit of the shifted operand, the shifted operand having a bit width of one of 8 bits and 16 bits; and
  - extending the bit width of the shifted operand to 32 bits.
- 20 4. The method of claim 3, wherein the method operation of extending the bit width of the shifted operand to 32 bits includes,

padding the extended bit width with a digital value.

5. The method of claim 4, wherein the digital value is selected from the group consisting of 1 and 0.

5

6. The method of claim 4, wherein the method operation of extending the bit width of the shifted operand to 32 bits includes,

determining the least significant bit of the shifted; and

repeatedly inserting the least significant bit to the shifted operand until the shifted

10 operand has been shifted by the shift value.

7. The method of claim 6, wherein the four stages include a two to one multiplexer at the first stage, the second stage and a third stage and a four to one multiplexer at a fourth stage.

15

8. A method for transparently presenting different size operands to be processed, comprising:

providing a first operand having a first bit-width;

determining a bit width of a second operand associated with a processor, the

20 second operand having a greater bit width than the first operand;

transforming the first operand by aligning a least significant bit of the first operand to a lowest bit position of a transformed operand, the transformed operand having a bit size equal to the second operand;

padding bits of the transformed operand; and

5 transmitting the transformed operand to the processor.

9. The method of claim 8, wherein the second operand is 32 bits wide.
10. The method of claim 8, wherein the first operand has a bit-width selected 10 form the group of 8 bits and 16 bits.
11. The method of claim 8, wherein the method operation of padding bits of the transformed operand includes,

filling higher bits of the transformed operand with a digital value, the higher bits 15 of the transformed operand excluding values associated with the first operand.
12. The method of claim 8, wherein the method operation of transforming the first operand by aligning a least significant bit of the first operand to a lowest bit position of a transformed operand includes,

sign extending the first operand by a value associated with the least significant bit. 20
13. The method of claim 8, further including:

processing the transformed operand;

transforming the processed operand to a post-processed operand having a first bit- 25 width; and

writing back the post-processed operand to a memory location.

14. A processor, comprising:

an input socket configured to receive and transmit data packets;

5 a transform unit for manipulating data, the transform unit configured to receive data packets from the input socket;

an arithmetic logic unit (ALU) configured to process multiple operands, the ALU including,

a shifter configured to shift each of the multiple operands in stages; and

10 an adder configured to add operands;

an alignment block having circuitry configured to align a lowest significant bit of the each of the multiple operands prior to transferring the each of the multiple operands to the ALU, the alignment block circuitry extending the each of the multiple operands to a defined bit width so that a pre-extending bit width of the each of the multiple operands

15 is transparent to the ALU; and

a bus connecting the input socket, the transform unit, the ALU, and the alignment block.

15. The processor of claim 14, wherein the ALU is a 32 bit ALU.

20

16. The processor of claim 14, wherein the shifter and the adder are supplied with parallel pathways.

17. The processor of claim 14, further including:

a second alignment block downstream from the ALU, the second alignment block configured to return each of the multiple operands received from the ALU to an original bit width.

5

18. The processor of claim 14, wherein the adder is a 32 bit adder.

19. The processor of claim 14, wherein the shifter includes four stages.

10 20. The processor of claim 19, wherein the shifter includes 3-two to one multiplexers and a four to one multiplexer, the 3-two to one multiplexers and the four to one multiplexer defining the four stages.